



OceanBase SQL

参考指南

版本：01

日期：2013.7.30

前 言

概述

本文档主要介绍OceanBase数据库支持的SQL语言、语法规则和使用方法等。

读者对象

本文档主要适用于：

- 开发工程师。
- 数据库管理工程师。

修订记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本。

目 录

1 OceanBase SQL 简介	5
1.1 支持语句	5
1.2 数据类型	5
1.3 浮点数	6
1.4 函数	7
1.5 内部表	7
2 数据定义语言	8
2.1 CREATE TABLE 语句	8
2.2 ALTER TABLE 语句	10
2.3 DROP TABLE 语句	11
3 数据操作语言	13
3.1 INSERT 语句	13
3.2 REPLACE 语句	14
3.3 UPDATE 语句	14
3.4 DELETE 语句	15
3.5 SELECT 语句	15
3.5.1 基本查询	15
3.5.2 JOIN 句法	19
3.5.3 UNION 句法	20
3.5.4 DUAL 虚拟表	22
3.5.5 SELECT ... FOR UPDATE 句法	22
4 事务处理	23
5 数据库管理语句	25
5.1 用户及权限管理	25
5.1.1 新建用户	25
5.1.2 删除用户	26
5.1.3 修改密码	26
5.1.4 修改用户名	27
5.1.5 锁定用户	27

5.1.6 用户授权	28
5.1.7 撤销权限	29
5.2 修改用户变量	30
5.3 修改系统变量	31
5.4 修改系统配置项	31
6 预备执行语句	34
7 其他 SQL 语句	36
8. SQL 优化	38
8.1. 执行计划	38
8.2 内部优化规则	38
8.2.1 主键索引	39
8.2.2 并发执行	39

1 OceanBase SQL 简介

SQL (Structured Query Language) 是一种组织、管理和检索数据库存储数据的计算机语言。由于 OceanBase 数据库完全兼容 MySQL 的网络协议，所以 OceanBase SQL 用户可以使用 MySQL 客户端、Java 客户端和 C 客户端连接 OceanBase。如果您熟悉 MySQL，那么就可以直接使用 OceanBase SQL 服务了。

1.1 支持语句

OceanBase SQL 语句中的关键字、表名、列名、函数名等均大小写不敏感。表名和列名都转换为小写之后存入 **schema** 中，所以即使用户建表时候列名是大写的，查询的时候获得的列名也是小写。

OceanBase SQL 语法遵循 SQL92 标准，单引号表示字符串；双引号表示表名、列名或函数名。双引号内可以出现 SQL 保留的关键字。

OceanBase 没有 Database 的概念，可以理解为一个 OceanBase 集群只有一个 Database，所以用户不需要也不能使用 “USE DATABASE” 语句来指定 Database。

目前版本支持的语句有 CREATE TABLE, DROP TABLE, ALTER TABLE, SELECT, INSERT, REPLACE, DELETE, UPDATE, SET, SHOW 等。

1.2 数据类型

OceanBase 数据库中没有字符集的概念。在数据库内部，char、varchar、binary 和 varbinary 都存储为不解释内容的二进制变长字符串，类似于 varbinary。这种类型的比较使用的是字节序。

此外，在数据库建表时定义的 varchar 列的最大长度也是不起作用的。例如 varchar(32)，实际上可以插入大于 32 字节的串。

目前 OceanBase 返回给客户端的数据类型如[表 1-1](#)所示

表 1-1 数据类型

数据类型	说明
tinyint/smallint/mediumint/int/integer/bigint	在 OceanBase 中，tinyint、smallint、mediumint、int、integer 和 bigint 无论语义还是实现都是等价的，存储为 8 字节有符号整型，即 int64_t。
float	表示 4 字节浮点数。
double/real	表示 8 字节浮点数。 在 OceanBase 中，double 和 real 等价，均存储为 double 类型。
datetime/timestamp	OceanBase 不支持 time 和 date 类型，只支持 datetime 和 timestamp 类型（这两者等价）。
bool	布尔类型，表示 true 或者 false。
varchar/char/binary/varbinary	字符串，使用单引号。 在 OceanBase 中，varchar、char、binary 和 varbinary 等价，均存储为 varchar 类型。
numeric	暂不支持。
decimal	暂不支持。
createtime	OceanBase 数据库提供的特殊的数据类型，用于记录本行数据第一次插入时的时间，由系统自动维护，用户不能直接修改。 该类型的列不能作为主键的组成部分。
modifytime	OceanBase 数据库提供的特殊的数据类型，用于记录本行数据最近一次被修改的时间，由系统自动维护，用户不能直接修改。 该类型的列不能作为主键的组成部分。

1.3 浮点数

在 MySQL 中，型如“1.2345”的字面量是作为 decimal 类型处理的，型如“1.2345e18”的字面量才作为浮点数处理。而目前版本的 OceanBase 中，两者都作为浮点数类型 double 处理。

在 MySQL 中，在做表达式运算时，两个整数类型相除，结果是 decimal 类型。而目前版本的 OceanBase 中，两者相除的结果为 double 类型。

1.4 函数

OceanBase 遵循 SQL 标准，不支持 MySQL COUNT(DISTINCT expr, expr, [expr...])的语法，即聚集函数中只能出现一个 value 表达式，不支持多个。例如：count(distinct c1, c2)。

OceanBase 目前支持的系统函数非常有限，只有 length、substr、cast、current_time、current_date、current_timestamp、trim、lower、upper、coalesce 等。

1.5 内部表

OceanBase 内部表都以“__”开头，普通用户表请不要使用这种格式的名字。原则上，OceanBase 开发人员保留在不同版本间增删内部表和修改它们 schema 的权利，所以普通用户和应用程序请不要依赖于这些表的任何内容。

内部表参数说明请参见《OceanBase 0.4 安装指南》的“[7.3 内部表参数说明](#)”章节。

我们可以使用 **show tables;**的命令来查看数据库中有哪些表，如[图 1-1](#)所示。

图 1-1 内部表

```
mysql> show tables;
+-----+
| table_name |
+-----+
| __first_tablet_entry |
| __all_all_column |
| __all_join_info |
| __all_client |
| __all_cluster |
| __all_server |
| __all_server_stat |
| __all_sys_config |
| __all_sys_config_stat |
| __all_sys_param |
| __all_sys_stat |
| __all_table_privilege |
| __all_trigger_event |
| __all_user |
| a |
| test |
+-----+
16 rows in set (0.00 sec)
```

2 数据定义语言

DDL（Data Definition Language）的主要作用是建立数据库基本组件的，例如建立表，删除表和修改表等。

OceanBase 支持的 DDL 主要有 CREATE TABLE，DROP TABLE 和 ALTER TABLE。

2.1 CREATE TABLE 语句

该语句用于在 OceanBase 数据库中创建新表。

* 格式

CREATE TABLE [IF NOT EXISTS] *table_name* (*column_name data_type* [NOT NULL | NULL] [DEFAULT *default_value*] [AUTO_INCREMENT], ..., PRIMARY KEY (*column_name1, column_name2...*)) [*table_options_list*];

- 使用“IF NOT EXISTS”时，即使创建的表已经存在，也不会报错，如果不指定时，则会报错。
- “data_type”请参见“1.2 数据类型”章节。
- NOT NULL，DEFAULT，AUTO_INCREMENT 用于列的完整性约束。在选项可以在 SQL 语句中出现，但目前尚未实现。
- “table_option_list”内容请参见[表 2-1](#)，各子句间用“,”隔开。

表 2-1 表选项

参数	含义	举例
EXPIRE_INFO	在 UpdateServer 中的动态数据和 ChunkServer 中的静态数据合并时，满足表达式的行会自动删除。 一般可用于自动删除过期数据。	EXPIRE_INFO = '\$SYS_DATE > c1 + #0-0-0 24:00:00#', 表示自动删除 c1 列的值比当前时间小 24 小时的行。

参数	含义	举例
TABLET_MAX_SIZE	这个表的 Tablet 最大尺寸，单位为字节，默认为 256MB。	TABLET_MAX_SIZE = 268435456
REPLICA_NUM	这个表的 tablet 复本数，默认值为 3。	REPLICA_NUM = 3
COMPRESS_METHOD	存储数据时使用的压缩方法名，目前提供的方法有以下三种： · none（默认值，表示不作压缩） · lzo_1_0 · snappy_1_0	COMPRESS_METHOD = 'none'
USE_BLOOM_FILTER	对本表读取数据时，是否使用 Bloom Filter。 · 0：默认值，不使用。 · 1：使用。	USE_BLOOM_FILTER = 0

注意：*OceanBase 内部数据以 b 树为索引，按照 Primary Key 排序，因此建表的时候，必须指定 Primary Key。Primary Key 有两种指定方式，详细请参见“举例”部分。*

* 举例

1. 执行以下命令，创建数据库表。
CREATE TABLE test (c1 int primary key, c2 varchar)
REPLICA_NUM = 3, COMPRESS_METHOD = 'none';
 或者
CREATE TABLE test (c1 int, c2 varchar, primary key(c1))
REPLICA_NUM = 3, COMPRESS_METHOD = 'none';
2. 执行命令查看表信息，如 [图 2-1](#) 所示。
SHOW tables;
DESCRIBE test;

图 2-1 CREATE TABLE

```
mysql> show tables;
+-----+
| table_name |
+-----+
| __first_tablet_entry |
| __all_all_column |
| __all_join_info |
| __all_client |
| __all_cluster |
| __all_server |
| __all_server_stat |
| __all_sys_config |
| __all_sys_config_stat |
| __all_sys_param |
| __all_sys_stat |
| __all_table_privilege |
| __all_trigger_event |
| __all_user |
| test |
+-----+
15 rows in set (0.00 sec)

mysql> DESCRIBE test;
+-----+-----+-----+-----+-----+-----+
| field | type          | nullable | key | default | extra |
+-----+-----+-----+-----+-----+-----+
| c1    | int           |          | 1   | NULL    |       |
| c2    | varchar(-1)   |          | 0   | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

2.2 ALTER TABLE 语句

该语句用于修改已存在的表的设计。

OceanBase 数据库目前只支持增加和删除列。

* 格式

增加列: ALTER TABLE *table_name* ADD [COLUMN] *column_name* *data_type*;

删除列: ALTER TABLE *table_name* DROP [COLUMN] *column_name*;

* 举例

1. 增加列前, 执行以下命令查看表信息, 如[图 2-2](#)所示。
DESCRIBE test;

图 2-2 ADD 前

field	type	nullable	key	default	extra
c1	int		1	NULL	
c2	varchar(-1)		0	NULL	

2 rows in set (0.00 sec)

2. 执行以下命令增加 c3 列。
ALTER TABLE test ADD c3 int;
3. 增加列后，执行以下命令查看表信息，如[图 2-3](#)所示。
DESCRIBE test;

图 2-3 ADD 后

field	type	nullable	key	default	extra
c1	int		1	NULL	
c2	varchar(-1)		0	NULL	
c3	int		0	NULL	

3 rows in set (0.00 sec)

4. 执行以下命令删除 c3 列。
ALTER TABLE test DROP c3;
5. 删除列后，执行以下命令查看表信息，如[图 2-4](#)所示。
DESCRIBE test;

图 2-4 DROP 后

field	type	nullable	key	default	extra
c1	int		1	NULL	
c2	varchar(-1)		0	NULL	

2 rows in set (0.00 sec)

2.3 DROP TABLE 语句

该语句用于删除 OceanBase 数据库中的表。

* 格式

DROP TABLE [IF EXISTS] *table_name*;

- 使用“IF EXISTS”时，即使要删除的表不存在，也不会报错，如果不指定时，则会报错。
- 同时删除多个表时，用“,”隔开。

* 举例

DROP IF EXISTS TABLE test;

3 数据操作语言

DML（Data Manipulation Language）的主要作用是根据需要写入、删除、更新数据库中的数据。

OceanBase 支持的 DML 主要有 INSERT，REPLACE，SELECT，UPDATE 和 DELETE。

3.1 INSERT 语句

该语句用于添加一个或多个记录到表。

OceanBase 中所有的表格都以主键为唯一索引，所以 INSERT 的行必须包含所有主键列的值，且主键列不能为 NULL。

* 格式

INSERT INTO *table_name* [(*column_name*,...)] VALUES (*column_values*,...);

- [(*column_name*,...)]用于指定插入数据的列。
- 同时插入多列时，用“,”隔开。

* 举例

1. 执行以下命令，插入行。
INSERT INTO test VALUES (1, 'hello alipay'),(2, 'hello ob');
2. 执行以下命令查看插入的行，如[图 3-1](#)所示。
SELECT * FROM test;

图 3-1 INSERT

```
+-----+-----+
| c1    | c2                |
+-----+-----+
|      1 | hello alipay      |
|      2 | hello ob          |
+-----+-----+
2 rows in set (0.01 sec)
```

3.2 REPLACE 语句

REPLACE 语句的语法和 INSERT 相同，语义有别：如果本行已经存在，则修改对应列的值为新值；如果不存在，则插入。

由于 OceanBase 系统架构的特点，REPLACE 语句的性能要显著优于 INSERT：REPLACE 语句的执行不需要向 Chunk Server 请求数据，而 INSERT 则需要。所以，如果你的应用程序在数据操作的语义上可以使用 REPLACE 或 INSERT，请优先使用 REPLACE 语句。

* 格式

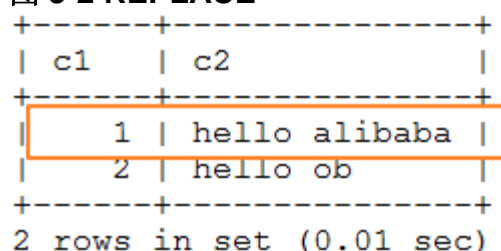
```
REPLACE INTO table_name [(column_name,...)] VALUES  
(column_values,...);
```

- [(*column_name*,...)]用于指定插入数据的列。
- 同时替换多列时，用用“,” 隔开

* 举例

1. 执行以下命令，替换表 test 中的行。
REPLACE INTO test VALUES (1, 'hello alibaba'),(2, 'hello ob');
2. 执行以下命令查看表中的行，如[图 3-2](#)所示。
SELECT * FROM test;

图 3-2 REPLACE



c1	c2
1	hello alibaba
2	hello ob

2 rows in set (0.01 sec)

3.3 UPDATE 语句

该语句用于修改表中的字段值。

OceanBase 数据库目前只支持指定 Primary Key 条件的单行更新，且暂不支持 DEFAULT。

* 格式

```
UPDATE table_name SET column_name1=column_values  
[column_name2=column_values] ... WHERE where_condition;
```

* 举例

1. 执行以下命令，修改“hello ob”为“hello oceanbase”。
UPDATE test SET c2 ='hello oceanbase' WHERE c1=2;
2. 执行以下命令查看修改后的信息。如[图 3-3](#)所示。
SELECT * FROM test;

图 3-3 UPDATE

c1	c2
1	hello alibaba
2	hello oceanbase

2 rows in set (0.00 sec)

3.4 DELETE 语句

该语句用于删除表中符合条件的行。

OceanBase 数据库目前只支持指定 Primary Key 条件的单行删除。

* 格式

DELETE FROM *table_name* **WHERE** *where_condition*;

注意：*where_condition* 必须指定关于 Primary Key 的条件。

* 举例

1. 执行以下命令删除“c1=2”的行。其中 c1 列为表 test 中的 Primary Key。
DELETE FROM test WHERE c1 = 2;
2. 执行以下命令查看删除行后的表信息，如[图 3-4](#)所示。
SELECT * FROM test;

图 3-4 DELETE

c1	c2
1	hello alibaba

1 row in set (0.01 sec)

3.5 SELECT 语句

该语句用于查询表中的内容。

3.5.1 基本查询

* 格式

SELECT [ALL | DISTINCT] *select_list* [AS *other_name*] FROM *table_name* [WHERE *where_conditions*] [GROUP BY *group_by_list*] [HAVING *search_confitions*] [ORDER BY *order_list* [ASC | DESC]] [LIMIT {[offset,] row_count | row_count OFFSET offset}];

SELECT 子句说明如[表 3-1](#)所示。

表 3-1 子句说明

子句	说明
ALL DISTINCT	在数据库表中，可能会包含重复值。指定“DISTINCT”，则在查询结果中相同的行只显示一行；指定“ALL”，则列出所有的行；不指定时，默认为“ALL”。
<i>select_list</i>	列出要查询的列名，用“,”隔开。也可以用“*”表示所有列。
AS <i>other_name</i>	为输出字段重新命名。
FROM <i>table_name</i>	必选项，指名了从哪个表中读取数据。
WHERE <i>where_conditions</i>	可选项，WHERE 字句用来设置一个筛选条件，查询结果中仅包含满足条件的数据。 <i>where_conditions</i> 为表达式。
GROUP BY <i>group_by_list</i>	用于进行分类汇总。
HAVING <i>search_confitions</i>	HAVING 字句与 WHERE 字句类似，但是 HAVING 字句可以使用累计函数（如 SUM，AVG 等）。
ORDER BY <i>order_list</i> [ASC DESC]	用来按升序（ASC）或者降序（DESC）显示查询结果。不指定 ASC 或者 DESC 时，默认为 ASC。

子句	说明
<code>[LIMIT {[offset,] row_count row_count OFFSET offset}]</code>	<p>强制 SELECT 语句返回指定的记录数。 LIMIT 接受一个或两个数字参数。参数必须是一个整数常量。</p> <ul style="list-style-type: none"> · 如果给定两个参数，第一个参数指定第一个返回记录行的偏移量，第二个参数指定返回记录行的最大数目。初始记录行的偏移量是 0(而不是 1)。 · 如果只给定一个参数，它表示返回记录行的最大数目，偏移量为 0。

* 举例

假设现有表 a 和表 b 如[图 3-5](#)所示。

图 3-5 表 a 和表 b

表a

id	name	num
1	a	100
2	b	200
3	a	50

ALL 和 **DISTINCT**：在数据库表中，可能会包含重复值。指定“**DISTINCT**”，则在查询结果中相同的行只显示一行；指定“**ALL**”，则列出所有的行；不指定时，默认为“**ALL**”。

SELECT name FROM a;

或者

SELECT ALL name FROM a;

```
+-----+
```

```
| name |
```

```
+-----+
```

```
| a    |
```

```
| b    |
```

```
| a    |
```

```
+-----+
```

```
3 rows in set (0.01 sec)
```

SELECT DISTINCT name FROM a;

```
+-----+
| name |
+-----+
| a    |
| b    |
+-----+
2 rows in set (0.01 sec)
```

AS: 为输出字段重新命名。

SELECT id, name, num/2 AS avg FROM a;

```
+-----+-----+-----+
| id  | name | avg  |
+-----+-----+-----+
| 1   | a    | 50   |
| 2   | b    | 100  |
| 3   | a    | 25   |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

WHERE: 用来设置一个筛选条件，查询结果中仅包含满足条件的数据。

SELECT id, name, num FROM a WHERE name = 'a';

```
+-----+-----+-----+
| id  | name | num  |
+-----+-----+-----+
| 1   | a    | 100  |
| 3   | a    | 50   |
+-----+-----+-----+
2 rows in set (0.01 sec)
```

GROUP BY: 用于进行分类汇总。以下语句是按“name”求“num”的总和。

SELECT id, name, SUM(num) FROM a GROUP BY name;

```
+-----+-----+-----+
| id  | name | SUM(num) |
+-----+-----+-----+
| 1   | a    | 150      |
| 2   | b    | 200      |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

HAVING: HAVING 字句与 WHERE 字句类似，但是 HAVING 字句可以使用累计函数（如 SUM，AVG 等）。以下语句是查询 num 总和小于 160 的行。

SELECT id, name, SUM(num) as sum FROM a GROUP BY name HAVING SUM(num) < 160;

```
+-----+-----+-----+
| id  | name | sum  |
+-----+-----+-----+
| 1   | a    | 150  |
+-----+-----+-----+
1 row in set (0.01 sec)
```

ORDER BY: 用来按升序 (ASC) 或者降序 (DESC) 显示查询结果。不指定 ASC 或者 DESC 时, 默认为 ASC。

SELECT * FROM a ORDER BY num ASC;

```
+-----+-----+-----+
| id    | name  | num   |
+-----+-----+-----+
|      3 | a     | 50    |
|      1 | a     | 100   |
|      2 | b     | 200   |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

SELECT * FROM a ORDER BY num DESC;

```
+-----+-----+-----+
| id    | name  | num   |
+-----+-----+-----+
|      2 | b     | 200   |
|      1 | a     | 100   |
|      3 | a     | 50    |
+-----+-----+-----+
3 rows in set (0.01 sec)
```

LIMIT: 强制 SELECT 语句返回指定的记录数。以下语句为从第 2 行开始, 返回两行。

SELECT * FROM a LIMIT 1,2;

```
+-----+-----+-----+
| id    | name  | num   |
+-----+-----+-----+
|      2 | b     | 200   |
|      3 | a     | 50    |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

3.5.2 JOIN 句法

JOIN 连接分为内连接和外连接。外连接又分为左连接、右连接和全连接。两个表联接后, 可以使用 **ON** 指定条件进行筛选。

目前, OceanBase 的 JOIN 不支持 USING 子句, 并且 JOIN 的连接条件中必须至少有一个等值连接条件。

假设现有表 a 和表 b 如[图 3-6](#)所示。

图 3-6 表 a 和表 b

表a	
id	name
1	a
2	b
3	c

表b	
id	colour
1	red
2	blue

内连接：结果中只包含两个表中同时满足条件的行。

SELECT a.id, a.name, b.colour FROM a INNER JOIN b ON a.id = b.id;

```
+-----+-----+-----+
| id    | name  | colour |
+-----+-----+-----+
| 1     | a     | red    |
| 2     | b     | blue   |
+-----+-----+-----+
2 rows in set (0.01 sec)
```

左连接：结果中包含位于关键字 LEFT [OUTER] JOIN 左侧的表中的所有行，以及该关键字右侧的表中满足条件的行。

SELECT a.id, a.name, b.colour FROM a LEFT OUTER JOIN b ON a.id = b.id;

```
+-----+-----+-----+
| id    | name  | colour |
+-----+-----+-----+
| 1     | a     | red    |
| 2     | b     | blue   |
| 3     | a     | NULL   |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

右连接：结果中包含位于关键字 [RIGHT] [OUTER] JOIN 右侧的表中的所有行，以及该关键字左侧的表中满足条件的行。

SELECT a.id, a.name, b.colour FROM a RIGHT OUTER JOIN b ON a.id = b.id;

```
+-----+-----+-----+
| id    | name  | colour |
+-----+-----+-----+
| 1     | a     | red    |
| 2     | b     | blue   |
+-----+-----+-----+
2 rows in set (0.01 sec)
```

全连接：结果中包含两个表中的所有行。

SELECT a.id, a.name, b.colour FROM a FULL OUTER JOIN b ON a.id = b.id;

```
+-----+-----+-----+
| id    | name  | colour |
+-----+-----+-----+
| 1     | a     | red    |
| 2     | b     | blue   |
| 3     | a     | NULL   |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

3.5.3 UNION 句法

UNION 操作符用于合并两个或多个 SELECT 语句的结果集。使用 UNION 需要注意以下几点：

- UNION 内部的 SELECT 语句必须拥有相同数量的列。列也必须拥有相似的数据类型。同时，每条 SELECT 语句中的列的顺序必须相同。
- 默认地，UNION 操作符选取不同的值。如果允许重复的值，请使用 UNION ALL。
- UNION 结果集中的列名总是等于 UNION 中第一个 SELECT 语句中的列名。

UNION 指令的目的是将两个或多个 SELECT 语句的结果合并起来。从这个角度来看，UNION 跟 JOIN 有些类似，因为这两个指令都可以由多个表格中撷取资料。但是 UNION 只是将两个结果联结起来一起显示，并不是联结两个表。

假设现有表 a 和表 c 如[图 3-7](#)所示。

图 3-7 表 a 和表 c

表a			表c		
id	name	num	xuhao	mingzi	shumu
1	a	100	4	c	150
2	b	200	5	b	200
3	a	50			

执行以下两个命令，比较 UNION 和 UNION ALL 的区别：

```
SELECT name, num FROM a UNION SELECT mingzi, shumu FROM c;
```

```
+-----+-----+
| name | num |
+-----+-----+
| a    | 50  |
| a    | 100 |
| b    | 200 |
| c    | 150 |
+-----+-----+
4 rows in set (0.01 sec)
```

```
SELECT name, num FROM a UNION ALL SELECT mingzi, shumu FROM c;
```

```
+-----+-----+
| name | num |
+-----+-----+
| a    | 100 |
| b    | 200 |
| a    | 50  |
| c    | 150 |
| b    | 200 |
+-----+-----+
5 rows in set (0.01 sec)
```

3.5.4 DUAL 虚拟表

DUAL 是一个虚拟的表，可以视为一个一行零列的表。当我们不需要从具体的表来取得表中数据，而是单纯地为了得到一些我们想得到的信息，并要通过 SELECT 完成时，就要借助一个对象，这个对象就是 DUAL。一般可以使用这种特殊的 SELECT 语法获得用户变量或系统变量的值。

当 SELECT 语句没有 FROM 子句的时候，语义上相当于 FROM DUAL，此时，表达式中只能是常量表达式。

* 格式

```
SELECT [ALL | DISTINCT] select_list [FROM DUAL [WHERE  
where_condition]] [LIMIT {[offset,] row_count | row_count OFFSET offset}];
```

参数说明请参见“3.2.1 基本查询”。

* 举例

执行以下命令，计算 2 和 3 的积。

```
SELECT 2*3;
```

或者

```
SELECT 2*3 FROM DUAL;
```

```
+-----+  
|  2*3  |  
+-----+  
|      6 |  
+-----+  
1 row in set (0.00 sec)
```

3.5.5 SELECT ... FOR UPDATE 句法

SELECT ... FOR UPDATE 可以用来对查询结果所有行上排他锁，以阻止其他事务的并发修改，或阻止在某些事务隔离级别时的并发读取。即使用 FOR UPDATE 语句将锁住查询结果中的元组，这些元组将不能被其他事务的 UPDATE，DELETE 和 FOR UPDATE 操作，直到本事务提交。

注意的是，目前 OceanBase 实现有如下限制：

- 必须是单表查询。
- Where 条件中必须限定单行。

例如：**SELECT * FROM a where id = 1 FOR UPDATE;**

4 事务处理

数据库事务(Database Transaction) 是指作为单个逻辑工作单元执行的一系列操作。事务处理可以确保除非事务性单元内的所有操作都成功完成，否则不会永久更新面向数据的资源。

* 格式

开启事务语句格式如下：

START TRANSACTION [WITH CONSISTENT SNAPSHOT];或者 **BEGIN [WORK];**

- **WITH CONSISTENT SNAPSHOT** 子句用于启动一个一致的读取。该子句的效果与发布一个 **START TRANSACTION**，后面跟一个来自任何 OceanBase 表的 **SELECT** 的效果一样。
- **BEGIN** 和 **BEGIN WORK** 被作为 **START TRANSACTION** 的别名受到支持，用于对事务进行初始化。**START TRANSACTION** 是标准的 SQL 语法，并且是启动一个 **ad-hoc** 事务的推荐方法。

提交当前事务语句格式如下：

COMMIT [WORK];

回滚当前事务语句格式如下：

ROLLBACK [WORK];

* 举例

假设现有表 **a** 如[图 4-1](#)所示。执行事务：将 **id** 为 3 的的 **name** 改为 **c**，并插入一行当前卖出 **a** 的记录。

图 4-1 表 a

表a

id	name	num	sell_date
1	a	100	2013-06-21 10:06:43
2	b	200	2013-06-21 13:07:21
3	a	50	2013-06-21 13:08:15

1. 依次执行以下命令开始执行事务。
START TRANSACTION;
UPDATE a SET name = 'c' WHERE id = 3;
INSERT INTO a VALUES (4, 'a', 30, '2013-06-21 16:09:13');
COMMIT;
2. 事务提交后，执行查看表 a 信息，如[图 4-2](#)所示。
SELECT * FROM a;

图 4-2 TRANSACTION

id	name	num	sell_date
1	a	100	2013-06-21 10:06:43
2	b	200	2013-06-21 13:07:21
3	c	50	2013-06-21 13:08:15
4	a	30	2013-06-21 16:09:13

4 rows in set (0.01 sec)

说明：在事务还没有 **COMMIT** 之前，您可以查看下本事务中的操作是否已经生效，比如可以在 **COMMIT** 之前，加一句 “**SELECT * FROM a;**”。不过结果肯定是没有生效，在事务还没有 **COMMIT** 之前，你之前做的操作是不可见的。如果你想回滚该事务的话，直接用 “**ROLLBACK**” 替代 “**COMMIT**”。

执行事务时，要注意以下几点：

- 我们采用 **datetime** 类型的值作为主键，在进行 **INSERT** 操作的时候，不能直接在 **SQL insert** 语句中采用 **currenttime()** 函数来获取时间作为主键列的值，比如 **INSERT INTO a VALUES(4, 'a', 30, currenttime())**，则会报错的。我们只能在 **INSERT** 之前，先用 **currenttime()** 函数获取当前时间，然后在 **INSERT** 语句中直接用该确定的时间值。
- 由于事务有一个超时时间，所以当我们手动输入事务中的多个 **SQL** 语句的时候，由于打字时间太长，会导致整个事务超时。解决方法是：修改当前 **session** 中的系统变量 **ob_tx_timeout**，使得该值尽可能大，而不至于导致超时。修改系统参数的方法请参见“5.4 修改系统配置项”。
- 目前在 **OceanBase** 的事务中执行 **SELECT** 语句，是不能读取当前事务中未提交的数据的。这是一个已知的功能 **BUG**。用户如果有这个需求，可以暂时用 **SELECT ... FOR UPDATE** 语句代替。

5 数据库管理语句

数据库管理包括用户及权限管理、修改用户变量、系统变量和系统配置。

5.1 用户及权限管理

数据库用户权限管理包括新建用户、删除用户、修改密码、修改用户名、锁定用户、用户授权和撤销授权等。

5.1.1 新建用户

CREATE USER 用于创建新的 OceanBase 用户。创建新用户后，可以使用该用户连接 OceanBase。

* 格式

CREATE USER 'user' [IDENTIFIED BY [PASSWORD] 'password'];

- 必须拥有全局的 CREATE USER 权限或对 “__all_user” 表的 INSERT 权限，才可以使用 CREATE USER 命令。
- 新建用户后，“__all_user” 表会新增一行该用户的表项，新建的用户没有任何权限。如果同名用户已经存在，则报错。
- 使用自选的 IDENTIFIED BY 子句，可以为账户给定一个密码。
- 此处密码为明文，存入 “__all_user” 表后，服务器端会变为密文存储下来。
- 同时创建多个用户时，用 “,” 隔开。

* 举例

1. 执行以下命令创建 “sqluser01” 和 “sqluser02” 用户。
CREATE USER 'sqluser01' IDENTIFIED BY '123456', 'sqluser02' IDENTIFIED BY '123456';
2. 执行以下命令查看创建的用户，如 [图 5-1](#) 所示。
SELECT user_name FROM __all_user;

图 5-1 创建用户

```
+-----+
| user_name |
+-----+
| admin     |
| sqluser01 |
| sqluser02 |
+-----+
3 rows in set (0.00 sec)
```

5.1.2 删除用户

DROP USER 语句用于删除一个或多个 OceanBase 用户。

* 格式

DROP USER 'user';

- 必须拥有全局的 CREATE USER 权限或对 “__all_user” 表的 DELETE 权限，才可以使用 DROP USER 命令。
- 成功删除用户后，这个用户的所有权限也会被一同删除。
- 同时删除多个用户时，用 “,” 隔开。

* 举例

执行以下命令，删除 “sqluser02” 用户。

DROP USER 'sqluser02';

5.1.3 修改密码

用于修改 OceanBase 登录用户的密码。

*格式

SET PASSWORD [FOR 'user' =] 'password';或者 ALTER USER 'user'
IDENTIFIED BY 'password';

- 如果没有 For user 子句，则修改当前用户的密码。任何成功登陆的用户都可以修改当前用户的密码。
- 如果有 For user 子句，或使用第二种语法，则修改指定用户的密码。必须拥有对 “__all_user” 表的 UPDATE 权限，才可以修改制定用户的密码。

* 举例

执行以下命令将 “sqluser01” 的密码修改为 “abc123”。

```
SET PASSWORD FOR 'sqluser01' = 'abc123';  
或者  
ALTER USER 'sqluser01' IDENTIFIED BY 'abc123';
```

5.1.4 修改用户名

用于修改 OceanBase 登录用户的用户名。

* 格式

```
RENAME USER 'old_user' TO 'new_user';
```

- 必须拥有全局 CREATE USER 权限或者对 __users 表的 UPDATE 权限，才可以使用本命令。
- 同时修改多个用户名时，用 “,” 隔开。

* 举例

1. 修改前，执行以下命令查看用户，如[图 5-2](#)所示。
SELECT user_name FROM __all_user;

图 5-2 修改前

```
+-----+  
| user_name |  
+-----+  
| admin    |  
| sqluser01 |  
+-----+  
2 rows in set (0.00 sec)
```

2. 执行以下命令修改用户名。
RENAME USER 'sqluser01' TO 'obsqluser01';
3. 修改后，执行以下命令查看用户，如[图 5-3](#)所示。
SELECT user_name FROM __all_user;

图 5-3 修改后

```
+-----+  
| user_name |  
+-----+  
| admin    |  
| obsqluser01 |  
+-----+  
2 rows in set (0.00 sec)
```

5.1.5 锁定用户

锁定或者解锁用户。被锁定的用户不允许登陆。

* 格式

锁定用户：ALTER USER 'user' LOCKED;

解锁用户：ALTER USER 'user' UNLOCKED;

必须拥有对 “__users” 表的 UPDATE 权限，才可以执行本命令。

* 举例

锁定用户：ALTER USER 'obsqluser01' LOCKED;

解锁用户：ALTER USER 'obsqluser01' UNLOCKED;

5.1.6 用户授权

GRANT 语句用于系统管理员授予 OceanBase 用户操作权限。

* 格式

GRANT *priv_type* ON *table_name* TO 'user';

- 给特定用户授予权限。如果用户不存在则报错。
- 当前用户必须拥有被授予的权限（例如，user1 把表 t1 的 SELECT 权限授予 user2，则 user1 必须拥有表 t1 的 SELECT 的权限），并且拥有 GRANT OPTION 权限，才能授予成功。
- 用户授权后，该用户只有重新连接 OceanBase，权限才能生效。
- 用 “*” 代替 *table_name*，表示赋予全局权限，即对数据库中的所有表赋权。
- 同时把多个权限赋予用户时，权限类型用 “,” 隔开。
- 同时给多个用户授权时，用户名用 “,” 隔开。
- *priv_type* 的如[表 5-1](#)所示。

表 5-1 权限类型

权限	说明
ALL PRIVILEGES	除 GRANT OPTION 以外所有权限。
ALTER	ALTER TABLE 的权限。
CREATE	CREATE TABLE 的权限。
CREATE USER	CREATE USER, DROP USER, RENAME USER 和 REVOKE ALL PRIVILEGES 的权限。

权限	说明
DELETE	DELETE 的权限。
DROP	DROP 的权限。
GRANT OPTION	GRANT OPTION 的权限。
INSERT	INSERT 的权限。
SELECT	SELECT 的权限。
UPDATE	UPDATE 的权限。
REPLACE	REPLACE 的权限。
SUPER	SET GLOBAL 修改全局系统参数的权限。

本用户自动拥有自己创建的对象（目前基本上只有表）。例如，用户 `user1` 创建了表 `t1` 和 `t2`，那用户 `user1` 应该自动就有对 `t1` 和 `t2` 的 `ALL PRIVILEGES` 及 `GRANT OPTION` 权限，不再需要额外去授权。

* 举例

执行以下命令给 “`obsqluser01`” 赋予所有权限。

GRANT ALL PRIVILEGES, GRANT OPTION ON * TO 'obsqluser01';

5.1.7 撤销权限

REVOKE 语句用于系统管理员撤销 OceanBase 用户的操作权限。

* 格式

REVOKE *priv_type* ON *table_name* FROM 'user';

- 用户必须拥有被撤销的权限（例如，`user1` 要撤销 `user2` 对表 `t1` 的 `SELECT` 权限，则 `user1` 必须拥有表 `t1` 的 `SELECT` 的权限），并且拥有 `GRANT OPTION` 权限。
- 撤销 “`ALL PRIVILEGES`” 和 “`GRANT OPTION`” 权限时，当前用户必须拥有全局 `GRANT OPTION` 权限，或者对权限表的 `UPDATE` 及 `DELETE` 权限。
- 撤销操作不会级联。例如，用户 `user1` 给 `user2` 授予了某些权限，撤回 `user1` 的权限不会同时也撤回 `user2` 的相应权限。

- 用 “*” 代替 *table_name*，表示撤销全局权限，即撤销对数据库中所有表的操作权限。
- 同时对用户撤销多个权限时，权限类型用 “,” 隔开。
- 同时撤销多个用户的授权时，用户名用 “,” 隔开。
- *priv_type* 的如[表 5-1](#)所示。

* 举例

执行以下命令撤销 “obsqluser01” 的所有权限。

REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'obsqluser01';

5.2 修改用户变量

用户变量用于保存一个用户自定义的值，以便于在以后引用它，这样可以将该值从一个语句传递到另一个语句。用户变量与连接有关，即一个客户端定义的用户变量不能被其它客户端看到或使用，当客户端退出时，该客户端连接的所有变量将自动释放。

* 格式

SET @var_name = expr;

- 用户变量的形式为 **@var_name**，其中变量名 *var_name* 可以由当前字符集的文字数字字符、“.”、“_”和“\$”组成。
- 每个变量的 *expr* 可以为整数、实数、字符串或者 NULL 值。
- 同时定义多个用户变量时，用 “,” 隔开。

* 举例

1. 执行以下命令设置用户变量。

```
SET @a=2, @b=3;
SET @c = @a + @b;
```

2. 执行以下命令，查看用户变量，如[图 5-4](#)所示。

```
SELECT @a, @b, @c;
```

图 5-4 用户变量

```
+-----+-----+-----+
| @a    | @b    | @c    |
+-----+-----+-----+
| 2     | 3     | 5     |
+-----+-----+-----+
1 row in set (0.00 sec)
```

5.3 修改系统变量

系统变量和 SQL 功能相关，存放在“__all_sys_param”表中，如 autocommit, tx_isolation 等。系统变量的参数说明请参见《OceanBase0.4 安装指南》的[“7.3.10 all_sys_param”](#)。

OceanBase 维护两种变量：

- 全局变量
影响 OceanBase 整体操作。当 OceanBase 启动时，它将所有全局变量初始化为默认值。修改全局变量，必须具有 SUPER 权限。
- 会话变量
影响当前连接到 OceanBase 的客户端。在客户端连接 OceanBase 时，使用相应全局变量的当前值对该客户端的会话变量进行初始化。设置会话变量不需要特殊权限，但客户端只能更改自己的会话变量，而不能更改其它客户端的会话变量。

*说明：*全局变量的更改不影响目前已经连接的客户端的会话变量，即使客户端执行 SET GLOBAL 语句也不影响。

* 格式

设置全局变量的格式：

```
SET GLOBAL system_var_name = expr; 或者 SET  
@@global.system_var_name = expr;
```

设置会话变量的格式：

```
SET [SESSION | @@session. | LOCAL | local. | @@]system_var_name =  
expr;
```

- 用正确的数据类型设置相应的系统变量的值。
- 您可以使用 **SHOW SESSION VARIABLES like 'system_var_name'**或 **SHOW SESSION VARIABLES like 'system_var_name'**查看变量值。

* 举例

执行以下命令，修改会话变量中的 SQL 超时时间。

```
SET @@session.ob_tx_timeout = 900000;
```

5.4 修改系统配置项

配置项存放在“__all_sys_config”表中，一般针对每一类 server 进行配置，一般影响某个 server 的行为，比如 MS 的线程池大小，RS 的负载均衡策略等，当然也可以通过指定 IP 对某个 server 单独进行配置。

* 格式

```
ALTER SYSTEM SET param_name = expr [COMMENT 'text'] [SCOPE = conf_scope] SERVER_TYPE = server_type [CLUSTER = cluster_id | SERVER_IP= 'server_ip' SERVER_PORT = server_port];
```

- 修改系统配置项说明如[表 5-2](#)所示。

表 5-2 子句说明

子句	说明
<i>param_name</i> = <i>expr</i>	配置项请参见《OceanBase0.4 安装指南》的“ 7.3.8 all sys config ”和“ 7.4 配置参数说明 ”。
COMMENT ' <i>text</i> '	可选，用于添加关于本次修改的注释。建议不要省略。
SCOPE = <i>conf_scope</i>	<p>SCOPE 用来指定本次配置项修改的生效范围。它的值主要有以下三种：</p> <ul style="list-style-type: none"> - MEMORY：表明只修改内存中的配置项，修改立即生效，且本修改在 server 重启以后会失效（目前暂时没有配置项支持这种方式）。 - SPFILE：表明只修改配置表中的配置项值，当 server 重启以后才生效。 - BOTH：表明既修改配置表，又修改内存值，修改立即生效，且 server 重启以后配置值仍然生效。 <p>说明：SCOPE 默认值为 BOTH。对于不能立即生效的配置项，如果 SCOPE 使用 BOTH 或 MEMORY，会报错。</p>
SERVER_TYPE = <i>server_type</i>	服务器类型， ROOTSERVER\UPDATESERVER\CHUNKSERVER\MERGESERVER。
CLUSTER = <i>cluster_id</i>	表明本配置项的修改正对指定集群的特定 server 类型，否则，针对所有集群的特定 server 类型。
SERVER_IP= ' <i>server_ip</i> ' SERVER_PORT = <i>server_port</i>	只修改指定 server 实例的某个配置项。

- 同时修改多个系统配置项时，用“,”隔开。

- 您可以使用 **SHOW PARAMETERS LIKE 'param_name';**命令，查看系统配置项的值。

*** 举例**

修改特定 Root Server 的线程数：

```
ALTER SYSTEM SET thread_count=30 COMMENT 'Modify by Bruce'  
SCOPE = SPFILE SERVER_TYPE = ROOTSERVER SERVER_IP=  
'10.10.10.2' SERVER_PORT= 1234;
```

6 预备执行语句

OceanBase 实现了服务器端的真正的 Prepared statement。用户先通过客户端发送一个预备语句把要执行的 SQL 数据操作语句发给服务器，服务器端会解析这个语句，产生执行计划并返回给客户端一个句柄（名字或者 ID）。随后，用户可以使用返回的句柄和指定的参数反复执行一个预备好的语句，省去了每次执行都解析 SQL 语句的开销，可以极大地提高性能。

由于目前 OceanBase SQL 引擎的优化工作还做的不够，SQL 解析并产生执行计划的过程效率不高。而使用预备执行语句，可以省掉这一过程，直接用已经预备好的执行计划和用户指定的参数执行语句，这样可以极大的提高性能。所以，我们强烈推荐应用尽可能多地使用预备执行语句。

* PREPARE 语句

PREPARE *stmt_name* FROM *preparable_stmt*;

- *preparable_stmt* 为 SQL 数据操作语句，预备好的语句在整个 SQL 会话期间可以使用 *stmt_name* 这个名字来执行。
- 数据操作语句（即 SELECT, REPLACE, INSERT, UPDATE, DELETE）都可以被预备执行。
- 在被预备的 SQL 语句中，可以使用问号（?）表明一个之后执行时才绑定的参数。问号只能出现在 SQL 语句的常量中。一个被预备的语句也可以不包含问号。

* EXECUTE 语句

EXECUTE *stmt_name* [USING @*var_name* [, @*var_name*] ...];

- 一个使用 PREPARE 语句预备好的 SQL 语句，可以使用 EXECUTE 语句执行。
- 如果预备语句中有问号指明的绑定变量，需要使用 USING 子句指明相同个数的执行时绑定的值。USING 子句后只能使用 SET 语句定义的用户变量。

* DEALLOCATE 语句

DEALLOCATE PREPARE *stmt_name*;

或者

DROP PREPARE *stmt_name*;

删除一个指定的预备语句。一旦删除，以后就不能再执行。

*** 举例**

依次使用 PREPARE 查询表 a 中 id=2 的行。

```
PREPARE stmt1 FROM SELECT name FROM a WHERE id=?;  
SET @id = 2;  
EXECUTE stmt1 USING @id;  
DEALLOCATE PREPARE stmt1;
```

7 其他 SQL 语句

主要介绍 SHOW、DESCRIBE 和 EXPLAIN 语句。

*** SHOW 语句**

SHOW 语句说明如[表 7-1](#)所示。

表 7-1 SHOW 语句说明

语句	说明
SHOW COLUMNS {FROM IN} <i>table_name</i> [LIKE ' <i>pattern</i> ' WHERE <i>expr</i>];	查看指定表的列的信息。
SHOW CREATE TABLE <i>table_name</i> ;	查看可以用来建立指定表格的建表语句。
SHOW TABLES [LIKE ' <i>pattern</i> ' WHERE <i>expr</i>];	查看数据库中存在哪些表。
SHOW [GLOBAL SESSION] VARIABLES [LIKE ' <i>pattern</i> ' WHERE <i>expr</i>];	查看全局或会话的系统变量，默认为当前会话。
SHOW PARAMETERS;	查看各个配置项在各个 server 实例上的值。
SHOW WARNINGS [LIMIT [offset,] row_count] SHOW COUNT(*) WARNINGS;	获得上一条语句执行过程中产生的“Warning”信息，不包含“Error”。
SHOW GRANTS;	查看当前用户的权限。

*** DESCRIBE 语句**

DESCRIBE *table_name* [*col_name* | wild];

或者

DESC *table_name* [*col_name* | wild];

这个语句等同于 SHOW COLUMNS FROM 语句。

* **EXPLAIN** 语句

```
EXPLAIN [VERBOSE] {select_stmt | insert_stmt | update_stmt | delete_stmt |  
delete_stmt};
```

这个语句可以输出 SELECT, INSERT, UPDATE, DELETE, REPLACE 等 DML 语句内部的物理执行计划。VERBOSE 模式也会输出逻辑执行计划。DBA 和开发人员可以根据 EXPLAIN 的输出来优化 SQL 语句。

8 SQL 优化

主要介绍 OceanBase 的 SQL 优化，提高 OceanBase 的执行效率。

8.1 执行计划

Explain 语句，可以查看一个 DML 语句的执行计划。执行计划是一个物理运算符组成的树状结构。常见的物理运算符有 TableScan, Filter, Sort, GroupBy, Join, Project 等。例如，执行 **EXPLAIN SELECT name, value1, value2 from __all_sys_config_stat WHERE name = 'location_cache_timeout';**语句，可以得到如下一个 PLAN：

```
Project(columns=[expr=[COL],expr=[COL],expr=[COL]])
TableRpcScan(rpc_scan==[COL|varchar:location_cache_timeout|EQ])
Project(columns=[expr=[COL],expr=[COL],expr=[COL]])
```

这个 plan 由两个物理运算符组成，Project 运算符负责投影和表达式计算，它的输入数据由下层的 TableRpcScan 提供。Columns 参数列出了三个表达式，表示这个投影操作会产生 3 列数据。expr=[COL]代表一个后缀表达式，这个表达式会产生一个 TABLE_ID 为 NULL，COLUMN_ID 为 65519 的 cell。这个 cell 的值根据后缀表达式[COL]运算后产生，这里这个表达式是一个列引用，就是取 TABLE_ID 为 12，COLUMN_ID 为 25 的数据值。

第二个物理运算符是 TableRpcScan，它实际上由 RpcScan 操作符实现。这个物理运算符实际上是在 Chunk Server 上执行。它读取 TABLE_ID 为 12 的表中通过 Project(columns=[expr=[COL],expr=[COL],expr=[COL]])中指定 COLUMN_ID 为 25,27,28 的 3 列数据。然后经过一个 Filter 操作符进行过滤，过滤条件即后缀表达式[COL|varchar:location_cache_timeout|EQ]，它表示过滤 TABLE_ID 为 12，COLUMN_ID 为 25 的 cell，等于 varchar:location_cache_timeout 的行。

8.2 内部优化规则

OceanBase 为 SQL 语句产生执行计划的时候，实现了一些基于规则的优化策略，这里简单描述一些目前实现的规则。

8.2.1 主键索引

OceanBase 表格中数据存储都是按照主键排序的。如果一个 **SELECT** 查询的 **WHERE** 条件中限定了主键的所有列，那么查询可以使用主键索引进行优化。假设有一个表 **t1**：

```
CREATE TABLE t1 (c1 int, c2 int, c3 int, c4 int, primary key(c1, c2, c3))
```

下面举例说明优化规则：

- 使用等值条件限定所有主键的查询，会使用 **MultiGet** 操作进行单行查询。如 “**SELECT * FROM t1 WHERE c1 = 1 and c2 = 2 and c3 = 3**”。注意，这里的等值条件必须写成列在等号左边，例如 **1 = c1** 则不满足本规则。
- 使用 **IN** 表达式限定所有主键的查询，会使用 **MultiGet** 操作进行多行查询。如 “**SELECT * FROM t1 WHERE (c1, c2, c3) IN ((1,2,3), (4,5,6))**”。
- 使用简单比较操作限定主键前缀列取值范围的查询，会转换为对某些特定 **Tablet** 的扫描。如 “**SELECT * FROM t1 WHERE c1 >= 1 and c1 < 10 and c2 > 100 and c2 <= 200 and c3 > 300**”，会转换为对主键范围(,) 所有 **tablet** 的扫描。
- 不满足上面三种情况的，则需要对整个表进行全表扫描。特别的，目前执行计划生产过程没有做表达式变化。所以 “**SELECT * FROM t1 WHERE (c1 = 1 and c2 = 2 and c3 = 3) OR (c1 = 10 and c2 = 20 and c3 = 30)**” 这个语句不会使用 **MultiGet** 优化，应用应该使用 **IN** 表达式执行这种多行查询。

8.2.2 并发执行

MS 向 **CS** 读取基本表数据的时候，是并发查询多个 **CS** 的。除此之外，如果满足某些条件，**MS** 会把聚合操作分发到拥有数据的相关 **CS** 上执行，然后把 **CS** 汇总后的结果再做汇总。我们把这个优化叫做“聚合操作下压”，它需要查询满足以下一些条件：

- 单表查询。
- 没有 **UNION**, **INTERSECT**, **EXCEPT** 等集合操作。
- 没有 **ORDER BY** 子句。
- 有 **GROUP BY** 子句，或者有聚集函数，且任何聚集函数不能有 **DISTINCT** 修饰符。
- 系统变量开关 **ob_group_agg_push_down_param** 为 **true**（默认值）。

此外，**limit** 操作也可以下压到 **CS** 端执行，以减少需要网络传输的数据。适用这个优化的查询需要满足一下条件：

- 单表查询。

- 没有 UNION, INTERSECT, EXCEPT 等集合操作。
- 没有 ORDER BY 子句。
- 没有 GROUP BY 子句以及聚集函数。
- 当然需要有 LIMIT 子句。